DUCLAUT Nathan
GARCIN Anthony
(07/12/2021)

Rapport de TP3 – Représentation visuelle d'objet

I. Introduction

L'objectif de ce TP est d'afficher dans une fenêtres plusieurs objets 3D qui forment une maison. Nous utiliserons pour cela les modules pygame et pyOpenGl. Nous pourrons observer la représentation sous différents angles en effectuant diverses rotations, translations et zooms.

II. Préparation

Question (1)

Le code ouvre et ferme immédiatement une fenêtre pygame. Le programme importe d'abord et initialise le module pygame. Ensuite, il affiche sur l'écran une fenêtre de taille 300*200 puis la quitte immédiatement.

Question (2)

Ce programme permet d'ouvrir une fenêtre pygame et de la fermer des lors que l'on appuie sur une touche du clavier. Ici, comme précédemment, le programme affiche à l'écran une fenêtre 300*200 puis une boucle tant que laisse cette fenêtre ouverte jusqu'à la pression d'une touche du clavier qui fait quitter cette boucle pour arriver à l'instruction quit.

III. Decouverte de l'environnement de travail du TP

Question (1)_a

La commande affiche un repère dans un fenêtre pygame

Nous nous intéressons également à la structure du programme. Le programme principal permet d'afficher les éléments 3D relatifs à chacune des questions. Les autres programmes sont des éléments nécessaires à la construction de la maison complète et sont composés d'une classe, d'un setter d'un getter et d'une fonction draw. Le setter initialise les valeurs et le getter récupèrent les valeurs passées en paramètre. Enfin, l'instruction draw sert à l'affichage.

Question (1)_c

Pour que l'axe z soit représenté verticalement, on utilise une matrice de rotation à l'aide de la commande glRotatef :

```
def initializeTransformationMatrix(self):
    gl.glMatrixMode(gl.GL_PROJECTION)
    gl.glLoadIdentity()
    glu.gluPerspective(70, (self.screen.get_width()/self.screen.get_height()), 0.1, 100.0)


    gl.glMatrixMode(gl.GL_MODELVIEW)
    gl.glLoadIdentity()
    gl.glTranslatef(0.0,0.0, self.parameters['screenPosition'])
    gl.glRotatef(-90,1,0,0)
```

IV. Mise en place des interactions avec l'utilisateur pygame

Question (1)_d

Comme dans la question précédente, nous utilisons la commande glRotatef pour ordonner la rotation lors de la pression sur les touches a et z du clavier.

```
def processKeyDownEvent(self):
    # Rotates around the z-axis
    if self.event.dict['unicode'] == 'Z' or (self.event.mod & pygame.KMOD_SHIFT and self.event.key == pygame.K_z):
        gl.glRotate(-2.5, 0, 0, 1)
    elif self.event.dict['unicode'] == 'z' or self.event.key == pygame.K_z:
        gl.glRotate(2.5, 0, 0, 1)

    # Draws or suppresses the reference frame
    elif self.event.dict['unicode'] == 'a' or self.event.key == pygame.K_a:
        self.parameters['axes'] = not self.parameters['axes']
        pygame.time.wait(300)
```

Question (1)_e

Ici, la molette nous permet de zoomer et dézoomer sur la fenêtre d'affichage. Nous utilisons la commande glScalef pour changer d'échelle.

```
def processMouseButtonDownEvent(self):
    if self.event.button == 4:
        gl.glScalef(1.25, 1.25, 1.25)

    elif self.event.button == 5:
        gl.glScalef(0.75, 0.75, 0.75)
```

Question (1)_f

Les boutons de souris doivent nous permettre de translater et tourner nos objets. On ajoute donc une condition pour chaque bouton et nous leur association des mouvements de rotations et de translations avec les commande glRotatef et glTranslatef :

```
def processMouseMotionEvent(self):
    if pygame.mouse.get_pressed()[0] == 1 :
      gl.glRotatef(self.event.rel[0],0, 1, 0)
      gl.glRotatef(self.event.rel[1], 1, 0, 0)
    elif pygame.mouse.get_pressed()[2] == 1 :
      gl.glTranslatef(self.event.rel[0]*0.05,0,0,-self.event.rel[1]*0.05)
```

V. Création d'une section

Question (2)_a

Nous construisons notre section en prenant pour origine [0,0,0]. Pour les arrêtes nous prenons les valeurs de hauteur, largeur et épaisseur rentrées en paramètres. Pour construire une face nous lui associons 4 arrêtes en fonction de leur position dans la définition de la fonction :

```
def generate(self):
    self.vertices = [
        [0,0,0],
        [0,0,self.parameters['height']],
        [self.parameters['width'],0,self.parameters['height']],
        [self.parameters['width'],0,0],
        [0,self.parameters['thickness'],0],
        [0,self.parameters['thickness'],self.parameters['height']],
        [self.parameters['width'],self.parameters['thickness'],self.parameters['height']],
        [self.parameters['width'],self.parameters['thickness'],0]
            ]
    self.faces = [
            [0,3,2,1],
            [0,4,5,1],
            [3,7,6,2],
            [4,7,6,5],
            [1,5,6,2],
            [0,4,7,3]
            ]
```

Question (2)_b/c

L'instruction **Configuration().add(section).display()** permet l'ajout d'objets de la classe section avec des données de la classe configuration.

On construit la fonction draw avec une matrice de projection de la section qui permettra l'affichage de celle-ci.

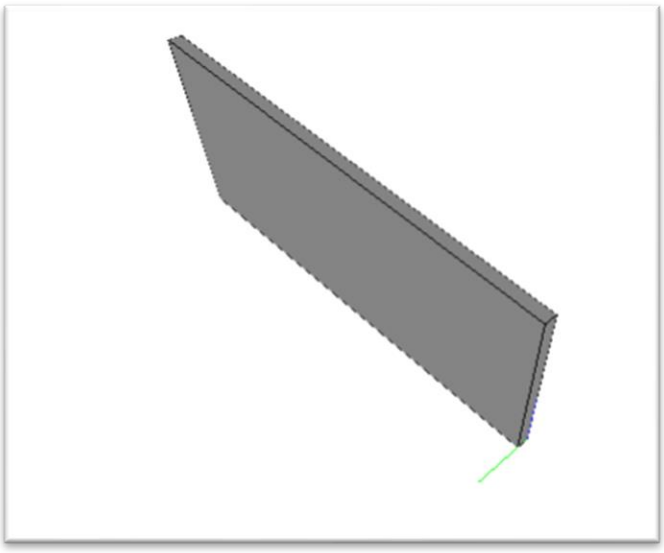
```
def draw(self):
    gl.glPushMatrix()
    if self.getParameter('edges') == True :
        self.drawEdges()
    for i in self.faces :
        gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_FILL) # on trace les faces : GL_FILL
        gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
        gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
        for j in range(4):
            gl.glVertex3fv(self.vertices[i[j]])
        gl.glEnd()
    gl.glPopMatrix()
```

Les lignes 3 et 4 correspondent à la question 2)c avec drawEgdes executé en premier via le getter.

La fonction précédente ne permet pas d'afficher les arrêtes de la section. Ainsi, on leurs donne le code couleur [0,0,0] du noir.

```
def drawEdges(self):
    for i in self.faces:
        gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE)
        gl.glBegin(gl.GL_QUADS)
        gl.glColor3fv([0,0,0])
        for j in range (4):
            gl.glVertex3fv(self.vertices[i[j]])
        gl.glEnd()
```

Finalement, on obtient bien un mur gris avec des arrêtes noirs.



VI. Création d'un mur

Question (3)_a

Nous reprenons la structure de la fonction draw précédente mais l'on utilise à présent la fonction draw de la classe section pour construire le mur.

```
# Draws the faces
def draw(self):
    gl.glPushMatrix() # Crée une matrice de projection temporaire
    gl.glRotatef(self.parameters['orientation'],0,0,1)
    self.parentSection.drawEdges() # Trace les arêtes
    for x in self.objects:
        x.draw()    # Appele la méthode draw de section
    gl.glPopMatrix() # Termine la matrice de projection temporaire
```

VII. Création d'une maison

Question (4)_a

De manière similaire, pour construire notre maison, on utilise cette fois la fonction draw de la classe wall pour construire une maison faite de murs.

```
# Draws the house
def draw(self):

    gl.glPushMatrix()
    gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1],self.parameters
    ['position'][2])
    gl.glRotatef(self.parameters['orientation'],0,0,1)
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
    for x in self.objects:


        x.draw() #On trace l'objet
        gl.glPopMatrix() # Termine la matrice de projection temporaire
```

On rajoute dans les caractéristiques de chacun des 4 murs et l'origine de la maison dans House.

```
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({'position': [0, 0, 0], 'width':7, 'height':2.6, 'orientation':0,'thickness':0.2})
    wall2 = Wall({'position': [7, 0.2, 0], 'width':7, 'height':2.6, 'orientation':90,'thickness':0.2})
    wall3 = Wall({'position': [0, 7.2, 0], 'width':7, 'height':2.6, 'orientation':0,'thickness':0.2})
    wall4 = Wall({'position': [0, 7.2, 0], 'width':7, 'height':2.6, 'orientation':-90,'thickness':0.2})
    house = House({'position': [-3, 1, 0], 'orientation':0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    return Configuration().add(house)
```